



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2013-0081784
(43) 공개일자 2013년07월18일

(51) 국제특허분류(Int. Cl.)

G06F 12/08 (2006.01)

(21) 출원번호 10-2012-0002781

(22) 출원일자 2012년01월10일

심사청구일자 2012년01월10일

(71) 출원인

고려대학교 산학협력단

서울 성북구 안암동5가 1

(72) 발명자

서보준

서울특별시 관악구 봉천동 관악현대아파트 121동 901호

김성우

서울특별시 성북구 안암동5가 16-2 A506

(74) 대리인

특허법인엠에이피에스

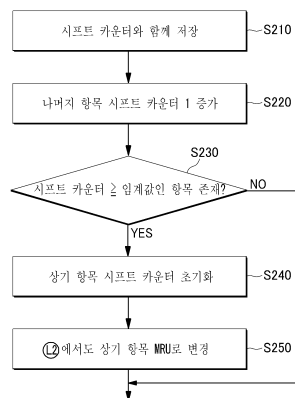
전체 청구항 수 : 총 7 항

(54) 발명의 명칭 계층 캐시 시스템 및 방법

(57) 요약

본 발명은 캐시 교체 방법으로 LRU(Least Recently Used) 방식을 사용하는 계층형 캐시 시스템을 운영하는 방법에 있어, 각 계층의 상위 메모리와 하위 메모리에 대하여, (a) 상기 상위 메모리가 요청된 데이터를 저장하고 있지 않을 경우, 상기 하위 메모리로부터 해당 데이터를 인출하여 상기 상위 메모리에 MRU(Most Recently Used) 상태로 저장한 후, 상기 상위 메모리가 저장하고 있는 나머지 데이터에 대해 시프트 카운터를 1 증가시키는 단계; (b) 상기 상위 메모리가 저장하고 있는 데이터 중 상기 시프트 카운터가 임계값 이상인 데이터가 있을 경우, 해당 데이터의 시프트 카운터를 초기화하는 단계; 및 (c) 상기 하위 메모리에서 상기 시프트 카운터가 초기화된 데이터에 대응되는 데이터를 MRU 상태로 변경하는 단계를 포함하는 것인 캐시 시스템 운영 방법을 제공한다.

대표도 - 도2



특허청구의 범위

청구항 1

캐시 교체 방법으로 LRU(Least Recently Used) 방식을 사용하는 계층형 캐시 시스템을 운영하는 방법에 있어,
각 계층의 상위 메모리와 하위 메모리에 대하여,

- (a) 상기 상위 메모리가 요청된 데이터를 저장하고 있지 않을 경우, 상기 하위 메모리로부터 해당 데이터를 인출하여 상기 상위 메모리에 MRU(Most Recently Used) 상태로 저장한 후, 상기 상위 메모리가 저장하고 있는 나머지 데이터에 대해 시프트 카운터를 1 증가시키는 단계;
- (b) 상기 상위 메모리가 저장하고 있는 데이터 중 상기 시프트 카운터가 임계값 이상인 데이터가 있을 경우, 해당 데이터의 시프트 카운터를 초기화하는 단계; 및
- (c) 상기 하위 메모리에서 상기 시프트 카운터가 초기화된 데이터에 대응되는 데이터를 MRU 상태로 변경하는 단계를 포함하는 것인 캐시 시스템 운영 방법.

청구항 2

제 1 항에 있어서,

상기 상위 메모리에 저장되어 있는 데이터는 상기 하위 메모리에도 저장되어 있는 것인 캐시 시스템 운영 방법.

청구항 3

제 1 항에 있어서,

상기 각 계층의 메모리는 멀티웨이 세트 연관 캐시 메모리(multi-way set associative cache memory)인 캐시 시스템 운영 방법.

청구항 4

제 3 항에 있어서,

상기 단계들은 상기 상위 메모리의 각 세트에 대해 독립적으로 수행되는 것인 캐시 시스템 운영 방법.

청구항 5

제 3 항에 있어서,

상기 (a) 단계는 상기 단계들을 수행할 대상 세트를 선택하는 단계를 포함하는 캐시 시스템 운영 방법.

청구항 6

제 3 항에 있어서,

상기 임계값은 상기 상위 메모리의 웨이 수를 상기 하위 메모리의 크기로 나눈 후 상기 값으로 상기 상위 메모리의 크기를 나눈 값에서 1을 뺀 값인 캐시 시스템 운영 방법.

청구항 7

제 1 항에 있어서,

상기 상위 메모리는 각 블럭별로 상기 시프트 카운터를 저장하는 버퍼를 포함하며, 상기 각 블럭당 버퍼의 크기는 상기 임계값에 대해 2를 밑으로 한 로그(Log)를 취한 값인 캐시 시스템 운영 방법.

명세서

기술분야

[0001] 본 발명은 캐시 메모리에 관한 것으로서, 보다 상세하게는 계층 캐시 시스템 및 방법에 관한 것이다.

배경 기술

[0002] 오늘날 대다수의 프로세싱 디바이스는 중앙 처리 장치와 주메모리간 속도 차이를 극복하기 위해 하나 이상의 캐시를 사용한다.

[0003] 많은 경우, L1, L2 캐시 등이 계층 구조를 이루고 있으며, 또한 각 계층의 캐시는 주소 매핑 방법으로 연관 사상(associative mapping) 방식을 사용하고 데이터 교체 알고리즘으로 LRU(Least Recently Used) 교체 방식류를 사용하는 경우가 많다. 또한 대부분 상위 메모리는 하위 메모리의 부분 집합을 이루도록 구성된다. 즉, 상위 메모리는 하위 메모리보다 용량이 작으며, 하위 메모리에 저장되어 있는 데이터의 일부분을 인출해와 저장하는 경우가 많다. 이러한 구조는 상위 메모리와 하위 메모리 사이의 일관성(coherence)을 유지하기가 용이하기 때문이다. 전술한 구조의 메모리 구조는 널리 알려진 공지 기술이므로, 자세한 설명은 생략한다.

[0004] 일관성 유지를 위하여, 하위 캐시에서 제거되는 데이터(즉, 하위 캐시가 꽉 찬 상태에서의 LRU 데이터)는 상위 캐시에서 가장 최근에 참조된 데이터(즉, MRU(Most Recently Used) 데이터)일 경우라도 제거되어야 한다. 상위 캐시는 하위 캐시에 저장되어 있지 않은 데이터를 저장할 수 없기 때문이다. 가장 최근에 참조된 데이터는 곧 다시 참조될 가능성이 많으므로, 이는 상위 캐시의 히트율(hit rate)을 떨어뜨려, 캐시 시스템의 성능을 낮춘다.

[0005] 이 문제를 해결하기 위해 여러 가지 방안을 생각해볼 수 있다. 예를 들어, 상위 캐시에 접근할 때마다 하위 캐시에 통보하여 하위 캐시에서도 MRU가 되게 하는 방안이 있다. 하지만 트래픽이 매우 증가하므로 실효성은 없다. 하위 캐시에서 제거할 때마다 먼저 상위 캐시에 확인하는 방안도 있지만, 이 역시 그 만큼 시간 지연이 발생하므로 성능이 떨어진다.

[0006] 캐시 일관성과 관련하여 한국등록특허 제1003218360000호("메모리 제어기 및 그 조작 방법")에는 공유 메모리 병렬 프로세싱 시스템에서 캐시 일관성을 유지하는 구성이 개시되어 있다.

[0007] 또한, 한국등록특허 제10-0326980호("다중-레벨 메모리 계층 구조를 포함하는 데이터 처리 시스템용 캐시 일관성 프로토콜")는 다수의 캐시와 다수의 프로세서를 포함한 시스템에서 캐시 일관성을 유지하는 구성이 개시되어 있다.

발명의 내용

해결하려는 과제

[0008] 본 발명은 전술한 계층 캐시 시스템에서 각 계층 메모리간 일관성 유지 문제를 해결하기 위한 것으로서, 그 목적은 성능이 저하되지 않는 캐시 시스템 및 캐시 운영 방법을 제공하는 것이다.

과제의 해결 수단

[0009] 상기와 같은 목적을 달성하기 위한 본 발명의 제 1 측면에 따른 캐시 교체 방법으로 LRU(Least Recently Used) 방식을 사용하는 계층형 캐시 시스템을 운영하는 방법은, 각 계층의 상위 메모리와 하위 메모리에 대하여, (a) 상기 상위 메모리가 요청된 데이터를 저장하고 있지 않을 경우, 상기 하위 메모리로부터 해당 데이터를 인출하여 상기 상위 메모리에 MRU(Most Recently Used) 상태로 저장한 후, 상기 상위 메모리가 저장하고 있는 나머지 데이터에 대해 시프트 카운터를 1 증가시키는 단계; (b) 상기 상위 메모리가 저장하고 있는 데이터 중 상기 시프트 카운터가 임계값 이상인 데이터가 있을 경우, 해당 데이터의 시프트 카운터를 초기화하는 단계; 및 (c) 상기 하위 메모리에서 상기 시프트 카운터가 초기화된 데이터에 대응되는 데이터를 MRU 상태로 변경하는 단계를 포함하는 것을 특징으로 한다.

발명의 효과

[0010] 본 발명은 계층 캐시 시스템 및 캐시 운영 방법에 있어, 캐시간 일관성 유지 문제로 발생하는 캐시 시스템의 성능 저하를 효과적으로 감소시킨다.

[0011] 구체적으로, 상위 캐시의 히트율(hit rate)이 저하되는 것을 방지하는 효과를 얻는다.

[0012] 또한 이러한 효과를 캐시간 트래픽(traffic)이나 처리 시간 지연(delay)을 크게 늘리지 않고도 달성한다.

[0013] 또한 간단한 연산을 사용하므로 구현에 많은 추가 요소가 필요하지 않으며, 시스템 자원을 많이 소모하지 않는다.

도면의 간단한 설명

[0014] 도 1, 2는 본 발명에 따른 캐시 캐시 운영 방법의 흐름을 도시함.

도 3, 4, 5는 본 발명에 따른 캐시 시스템의 일실시예를 도시함.

도 6은 종래 기술의 캐시 운영 방법에 따른 데이터 입출력의 일실시예를 도시함.

도 7은 본 발명의 캐시 운영 방법에 따른 데이터 입출력의 일실시예를 도시함.

발명을 실시하기 위한 구체적인 내용

[0015] 아래에서는 첨부한 도면을 참조하여 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자가 용이하게 실시할 수 있도록 본 발명의 실시예를 상세히 설명한다. 그러나 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며 여기에서 설명하는 실시예에 한정되지 않는다. 그리고 도면에서 본 발명을 명확하게 설명하기 위해서 설명과 관계없는 부분은 생략하였으며, 명세서 전체를 통하여 유사한 부분에 대해서는 유사한 도면 부호를 붙였다.

[0016] 명세서 전체에서, 어떤 부분이 다른 부분과 "연결"되어 있다고 할 때, 이는 "직접적으로 연결"되어 있는 경우뿐 아니라, 그 중간에 다른 소자를 사이에 두고 "전기적으로 연결"되어 있는 경우도 포함한다. 또한 어떤 부분이 어떤 구성요소를 "포함"한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성요소를 제외하는 것이 아니라 다른 구성요소를 더 포함할 수 있는 것을 의미한다.

[0017] 본 발명에 따른 캐시 시스템은 계층 구조를 가진다. 상위 메모리는 하위 메모리의 부분 집합이다. 예를 들어, 최하위 캐시(LLC: Last Level Cache)는 주메모리(예: RAM)에 저장되어 있는 데이터 중 일부를 저장하며, 최상위 캐시인 L1 캐시는 자신의 하위 캐시인 L2 캐시에 저장되어 있는 데이터 중 일부를 저장한다. 계층 수준의 수는 제한이 없지만, 바람직한 일실시예에서 L2 캐시가 최하위 캐시가 되도록 구성될 수 있다. 많은 경우 L1 캐시는 데이터를 위한 DL1과 명령어를 위한 IL1으로 구분하여 구성되지만, 여기서는 DL1과 IL1을 구분하지 않고 L1으로 기술한다.

[0018] 또한 본 개시의 도면 및 상세 설명에서 L1과 L2는 어떤 계층에서의 상위 메모리, 하위 메모리를 대표하기 위해 사용된 식별자로, 각각 실제 L1과 L2에 한정되는 것이 아니다. 한 수준 더 상위 메모리 및 한 수준 더 하위 메모리도 각각 CPU 및 주메모리로 지칭하였다.

[0019] 각 계층의 캐시는 세트 연관 사상(set-associative mapping) 방식으로 주소를 매핑한다. 즉, 각 캐시는 멀티웨이 세트 연관 캐시 메모리(multi-way set associative cache memory)이며, 상위 캐시와 하위 캐시의 세트(set) 및 웨이(way) 수는 서로 독립적이다. 예를 들어, L1 캐시는 4-웨이 세트 연관 캐시, L2 캐시는 8-웨이 세트 연관 캐시로 구성될 수 있다.

[0020] 각 계층의 캐시는 LRU(Least Recently Used) 방식으로 분류되는 교체 방식(예: LRU, 의사-LRU(Pseudo-LRU) 등)을 사용하여 블록을 교체한다. 즉, 더이상 저장할 공간이 없어 하위 메모리로부터 인출해온 데이터를 저장하기 위해서 저장된 데이터 중 하나를 제거해야 하는 경우, 각 계층의 캐시는 가장 이전에 참조된(LRU: Least Recently Used) 블록을 제거한다. 데이터의 시간적 지역성(temporal locality)을 고려할 때, LRU 블록이 가까운 미래에 다시 참조될 가능성이 가장 낮기 때문이다.

[0021] 반면, 가장 최근에 참조된(MRU: Most Recently Used) 블록은 곧 다시 참조될 가능성이 가장 높으므로 계속 유지하는 것이 합리적임에도, 전술한 바와 같이 종래 기술에서는 하위 캐시와의 일관성을 유지하기 위해 이를 제거해야 하는 경우가 있다. 따라서 상위 캐시의 히트율(hit rate)이 낮아지므로 캐시 시스템의 성능이 저하된다.

[0022] 이 문제를 해결하기 위해 본 발명에 따른 캐시 시스템은 상위 캐시의 각 블록이 자신의 미스(miss) 횟수를 기록하게 한다. 자세한 내용은 도 1 및 도 2를 참조하여 설명한다.

[0023] 도 1은 본 발명에 따른 캐시 운영 방법의 전체적 흐름을 도시하고 있다.

[0024] 처리 장치(예: CPU, GPU 등) 또는 자신보다 상위 메모리로부터 데이터를 요청 받으면(S102), L1은 나머지 단계를 수행할 대상 세트를 선택한다(S104). 자세한 설명은 후술하겠지만, 일단 여기서는 해당 세트가 큐(Queue)로 동작하고 있다고 생각하고 설명을 진행하겠다.

- [0025] 선택된 세트에 요청된 데이터가 있는지 판단하여(S106), 있으면(hit) 해당 블록을 MRU로 변경하고(S108), 없으면(miss) L2에 요청을 전달한다(S110).
- [0026] L2도 자신의 상위 메모리와 같은 동작들을 수행하여, 세트를 선택하고(S112) 데이터가 있는지 확인한 후(S114), 있으면 MRU로 변경하고(S116), 없으면 자신의 하위 메모리에 요청을 전달한다(S118).
- [0027] 다음, 자신의 하위 메모리로부터 요청한 데이터가 반환되어 오면(S120), L2는 이를 저장할 공간이 있는지 확인하고(S122), 저장할 공간이 없으면 LRU를 제거하여 공간을 확보한 후(S124), 해당 데이터를 MRU로 저장한다(S126).
- [0028] 다음, L1도 자신의 하위 메모리와 같은 동작들을 수행하여, L2로부터 반환되어 온 데이터를(S128) 저장할 공간이 있는지 확인하고(S130), 저장할 공간이 없으면 LRU를 제거하여 공간을 확보한 후(S132), 해당 데이터를 MRU로 저장한다(S134).
- [0029] 다음, L1은 처리 장치 또는 자신의 상위 메모리에 요청받은 데이터를 반환한다(S136).
- [0030] L1이 데이터를 요청받고(S102) 이에 응답하여 요청받은 데이터를 반환하기(S136)까지의 단계 수를 보면, L1에 해당 데이터가 있을 때와 없을 때 처리 시간에 큰 차이가 있음을 알 수 있다. 단순히 단계 수만 세어보아도, L1 히트일 때는 1 단계만 거치면 되지만, L1 미스일 때는 최악의 경우 14 단계가 필요하다. 즉, 상위 메모리의 히트율이 캐시 시스템의 성능에 큰 영향을 미치므로, 상위 메모리가 곧 다시 참조될 가능성이 높은 데이터를 되도록 많이 유지하도록 하는 것이 중요하다.
- [0031] 그림에도 불구하고 전술한 바와 같이, 종래 기술에서는 상기 L2에서의 LRU 제거 단계(S124)에서 해당 데이터가 L1에도 존재하는지 확인하고 존재하면 L1과 L2의 일관성 유지를 위해 L1에서도 이를 제거하는 과정을 추가로 거쳐야 한다. 이 때문에 종래 기술은 곧 다시 참조될 가능성이 가장 높은 MRU 데이터를 제거해야 하는 경우에 캐시 시스템의 성능이 낮아진다는 문제를 갖고 있다. 또한 L2에서 LRU를 제거할 때마다 L1을 참조하는 과정을 거쳐야 하므로, 그에 따른 처리 지연(delay)과 트래픽(traffic) 증가에 의해 성능은 더욱 낮아진다.
- [0032] 이제 도 1의 단계들 중 상기 단계(S134)를 구체적으로 도시한 도 2를 통해 본 발명이 이 문제를 어떻게 해결하는지 살펴보자.
- [0033] 본 발명에 따른 캐시 운영 방법은 종래 기술과 달리, 각 블록에 대한 시프트 카운터를 함께 저장함으로써, 상위 메모리와 하위 메모리의 일관성을 유지하기 위해 상위 메모리의 히트율을 희생시켰던 문제를 해결한다.
- [0034] L1이 L2로부터 반환받은 데이터를 MRU로 저장할 때, 시프트 카운터도 함께 저장되며(S210), 저장되어 있는 나머지 항목들에 대한 시프트 카운터가 각각 1씩 증가된다(S220). 이 시프트 카운터가 임계값(L2 큐 길이-1)을 넘어서는 블록이 존재하는지 확인하고(S230), 존재할 경우 해당 블록에 대한 시프트 카운터를 초기화한 후(S240), L2에도 통보하여 L2에서도 상기 항목을 MRU로 변경하게 한다(S250).
- [0035] 따라서 MRU임에도 제거해야 하는 일이 발생하지 않아 캐시 시스템의 성능 저하를 막을 수 있다. 더구나 본 발명에 따른 캐시 운영 방법은 각 계층간 트래픽을 늘리지 않고도 L1의 히트율 감소를 막고 있다. 또한 카운터를 1 증가시키는 단순한 연산만을 필요로 하며 다른 작업과 병렬로 처리 가능하므로 처리 지연 또한 크게 늘어나지 않는다.
- [0036] 자세한 내용은 실시예를 통해 후술한다.
- [0037] 도 3, 4, 5는 본 발명에 따른 캐시 시스템의 일실시예를 도시하고 있다.
- [0038] 도 3에는 다양한 실시예의 사상이 도시되어 있다. 여기에서 중요한 것은 L1, L2를 L1 입장에서 큐(Queue)로 나타내는 데 필요한 사상이다. LRU 교체 알고리즘 자체가 큐를 기반으로 하고 있기 때문이다. 이러한 관점에서 중요하지 않거나 유효하지 않은 데이터는 '...'로 나타내었다. 예를 들어, L2의 웨이 수나 블록 크기는 하위 메모리와 연계하는데 있어서 필요한 것이므로 여기서는 중요하지 않다. 반면 관심 대상인 L1, L2, 메모리의 용량, L1 블록 크기 및 L1의 웨이 수에는 각각 x , y , z , p 및 n 이라는 변수를 지정하도록 하자.
- [0039] 먼저, L1의 큐 개수와 큐 길이를 구해보자. 일반적으로 멀티 웨이 세트 연관 캐시는 각 세트별로 LRU인 웨이를 결정하므로(후술하는 실시예 참조), L1의 큐 개수는 세트 수, 큐 길이는 웨이 수임을 쉽게 알 수 있다. 즉, L1을 큐로 나타낸다면, L1은 큐의 길이가 웨이 수인 큐가 세트 수 만큼 있다고 볼 수 있다.
- [0040] 이때 주의할 점은 여기서의 큐의 길이는 큐에 들어가는 항목의 개수를 말하는 것이지, 큐의 실제 크기가 아니라

는 것이다. 예를 들어, 1바이트 크기의 항목이 4개 들어가는 큐와 4바이트 크기의 항목이 4개 들어가는 큐는 모두 큐의 길이가 4이다. 도 3에 도시된 다양한 실시예 중 마지막 두 실시예를 비교해보면, 큐가 저장할 수 있는 항목의 개수는 동일하지만, 큐의 크기는 서로 다를 것임을 쉽게 알 수 있을 것이다.

[0041] L1의 세트 수는 전술한 사양으로부터 쉽게 도출된다. 총 용량 x 를 웨이 수 n 과 블록 크기 p 로 나누면 된다. 즉, L1 캐시는 길이가 웨이 수, 즉 n 만큼인 큐가 세트 수, 즉 $x/(n*p)$ 만큼 있는 메모리이다.

[0042] 다음, L1과의 관계에서 봤을 때 L2의 큐의 개수는 L1의 큐의 개수와 동일할 것임을 쉽게 이해할 수 있다. L1 큐에 없는 데이터를 L2 큐에서 가져오는 계층 구조이므로 각자의 큐가 서로 대응될 것이기 때문이다. 따라서 L2의 큐의 개수도 $x/(n*p)$ 개이다. 참고로, L2와 L2의 하위 메모리와의 관계에서는 전술한 내용에 따라 L2의 세트 수가 L2의 큐의 개수가 되고, L2의 웨이 수가 L2의 큐의 길이가 된다. 이때는 L2가 상위 메모리의 입장에 있게 되기 때문이다.

[0043] L2는 큐의 개수는 L1과 같은데 용량은 L1보다 큰 메모리이므로, L2를 구성하는 각 큐의 길이는 L1을 구성하는 각 큐의 길이보다 길 것임은 자명하다. L2 큐의 길이는 L2에 L1의 큐 하나가 몇 개 들어갈 수 있는나로 계산할 수 있다. 즉, L2를 L1의 큐당 크기로 나누면 된다. L1의 큐당 크기는 L1의 총 용량 x 에서 L1의 웨이 수 n 을 나누어 도출할 수 있으므로, L2의 큐의 길이는 L2의 총 용량 y 에서 L1의 큐당 크기인 x/n 을 나눈 값 즉, $(n*y)/x$ 가 된다.

[0044] 정리하면, 상위 메모리와 하위 메모리의 큐의 개수는 상위 메모리의 세트 수와 같고, 상위 메모리의 각 큐는 길이가 상위 메모리의 웨이 수이고, 하위 메모리의 각 큐는 길이가 하위 메모리의 총 용량을 상위 메모리의 총 용량으로 나눈 값에 상위 메모리의 웨이 수를 곱한 값이다.

[0045] L1과 L2의 길이가 위와 같이 도출되는 이유를 다른 식으로 설명하자면, L1의 각 큐의 길이가 n 인 이유는 L2의 어떤 블록이 L1에 들어갈 때 선택할 수 있는 곳이 n 개이기 때문이며, L2의 각 큐의 길이가 $(n*y)/x$ 인 이유는 L1의 어떤 블록에는 $(n*y)/x$ 개의 L2 블록이 대응될 수 있기 때문이다.

[0046] 여기서 본원 발명의 관심은 L2 큐의 길이이다. 본원 발명은 L1과 L2간 메시지 교환없이도 L1에서는 LRU가 아닌 L2에서 맨 끝까지 밀려나 제거될 위기에 처한 블록을 구하는 것이기 때문이다. 즉, L1이 각 블록이 L2에서 LRU인가 여부를 계산만으로 알아내는 것이 중요하다. LRU는 큐의 맨 끝에 위치해 있으므로 이러한 계산에는 큐의 길이가 필요하다. 즉, 도 2에서 설명한 바와 같이 본원 발명은 L1의 블록이 뒤로 밀리는 횟수, 즉 L1 시프트 카운터를 세어 이 값이 임계값을 넘어서는지 여부를 확인하는데, 이 임계값은 L2 큐 크기보다 하나 작은 값 즉, $(n*y)/x-1$ 이다.

[0047] 이제 도 3에 도시된 실시예 중 마지막 실시예를 사용하여 캐시 시스템의 동작을 살펴보자.

[0048] 도 4 및 도 5에 이러한 본 실시예의 구조 및 큐로 나타내었을 때의 모습이 각각 도시되어 있다.

[0049] 전술한 바와 같이 여기에서 L1, L2, 및 주메모리는 각각 어느 계층에서의 상위 메모리, 하위 메모리, 및 그 하위 메모리의 하위 메모리를 가리키기 위해 사용되었음을 주의하기 바란다.

[0050] 설명을 간단하기 위해 본 실시예의 L1, L2 캐시 및 주메모리는 매우 작은 용량을 갖고 있다. 각각 4바이트, 8바이트, 및 16바이트밖에 되지 않는다. L1, L2, 및 메모리의 블록 크기는 각각 1바이트이고, L1은 2-웨이 세트 연관 캐시 메모리이다.

[0051] 전술한 설명을 통해, 본 실시예의 L1의 세트 수는 2개이며, 따라서 L1 및 L2의 큐의 개수는 2개임을 알 수 있다. L1의 큐의 길이는 2개이고, L2의 큐의 길이는 4개이다. 전술한 바와 같이 큐의 길이는 저장할 수 있는 항목 수를 의미하지만, 본 실시예에서는 L1과 L2의 각 블록의 크기가 각각 1바이트이므로, 각 큐의 용량 또한 2바이트, 4바이트이다. L2의 큐의 길이가 4이므로, L1의 시프트 카운터가 가질 수 있는 최대값은 L2 큐의 길이에서 1을 뺀 값인 3이다.

[0052] 먼저 도 4를 참조하면, L1은 <웨이 0>와 <웨이 1>로 구성되어 있고, 각 웨이는 2개의 세트로 구성되어 있다.

[0053] L1과 L2, L2와 주메모리를 대응시키는 방법의 일실시예로 여기서는 주메모리(여기서의 주메모리는 실제 주메모리를 지칭)의 주소를 L1의 웨이 수인 2로 나눈 나머지를 사용하여 L1이나 L2의 어느 블록에 넣을 것인지를 결정하도록 하자. 즉, CPU 등 처리 장치나 L1의 상위 메모리로부터 데이터 요청이 들어오면, 해당 요청이 나타내는 주메모리 주소를 L1의 웨이 수인 2로 나눈 나머지가 짝수이면 L1 및 L2의 짝수 번지에, 홀수이면 L1 및 L2의 홀수 번지에 넣도록 하자. 만약 L1 캐시가 4-웨이 세트 연관 메모리라면 2 대신 4로 나눈 나머지를 사용하면

된다. 이는 주소 대응의 일실시예이며, 주소 대응 방식이 여기에 한정되는 것은 아니다.

- [0054] 편의상 L1, L2, 주메모리에서 각 블록을 주소를 사용하여 <블록 0>, <블록 1> 등으로 나타내면, 예를 들어, 주메모리의 주소가 15인 <블록 15>에 저장된 데이터를 요청받았을 때, $15\%2=1$ 이므로, 즉 15를 2로 나눈 나머지는 1이므로(%는 나머지를 구하는 연산자), 이 데이터는 L2에서는 <블록 1>, <블록 3>, <블록 5>, <블록 7> 중 하나, L1에서는 <블록 1>, <블록 3> 중 하나에 들어갈 수 있다. 이 데이터가 들어갈 수 있는 블록의 주소가 모두 홀수임을 주목하자.
- [0055] 이렇게 각 블록의 주소가 짝수와 홀수로 나누어지므로, 편의상 L1의 첫번째 세트는 <짝수 세트>, 두번째 세트는 <홀수 세트>라고 하자. 즉, <블록 0>는 <웨이 0>의 <짝수 세트>이고, <블록 1>는 <웨이 0>의 <홀수 세트>이며, <블록 2>는 <웨이 1>의 <짝수 세트>이고, <블록 3>는 <웨이 1>의 <홀수 세트>이다.
- [0056] 즉, 각 계층 메모리는 짝수 주소의 블록끼리, 홀수 주소의 블록끼리 나뉘어져 계층간 대응되므로, L1의 각 <짝수 세트>와 <홀수 세트>는 독립적으로 동작한다. 이제 전술한 설명에서 L1의 큐의 개수가 세트의 수와 같으며, 큐의 길이가 웨이의 수와 같다고 한 것이 쉽게 이해될 것이다.
- [0057] 예를 들어, L2의 첫번째 블록인 <블록 0>은 L1의 <짝수 세트>에 해당하는 <웨이 0> 또는 <웨이 1>, 즉 L1의 <블록 0> 또는 <블록 2>에 대응되며, L2의 <블록 1>은 L1의 <홀수 세트>에 해당하는 <웨이 0> 또는 <웨이 1>, 즉 L1의 <블록 1> 또는 <블록 3>에 대응된다. L2의 <블록 2>은 다시 L1의 <짝수 세트>에 해당하는 <웨이 0> 또는 <웨이 1>, 즉 L1의 <블록 0> 또는 <블록 2>에 대응되고, L2의 <블록 3>은 L1의 <홀수 세트>에 해당하는 <웨이 0> 또는 <웨이 1>, 즉 L1의 <블록 1> 또는 <블록 3>에 대응된다.
- [0058] 따라서, L1과 L2를 도 5와 같이 큐로 나타낼 수 있다.
- [0059] L1의 <짝수 세트>와 <홀수 세트>가 각각 별도의 큐가 되며, L2도 이와 대응되는 2개의 큐로 구성된다. 각각 <짝수 큐> 및 <홀수 큐>라고 하자.
- [0060] 물론 L1과 L2를 큐로 나타내는 것은 이해를 위해서일 뿐임은 당업자라면 쉽게 이해할 수 있을 것이다. 큐는 맨 왼쪽이 MRU, 맨 오른쪽이 LRU로 도시되지만, 실제로는 세트 내에서 주소가 낮은 블록이 MRU, 주소가 높은 블록이 LRU인 식으로 동작하는 것이 아니다. 예를 들어, L1의 <짝수 큐>에서 <블록 0>가 MRU, <블록 2>가 LRU 데이터를 저장하고 있을 수도 있지만, <블록 2>가 MRU, <블록 0>가 LRU 데이터를 저장하고 있을 수도 있다.
- [0061] 예를 들어 보자. 전술한 주메모리의 <블록 15>에 저장되어 있는 데이터가 예컨대 "HELLO"라고 하자. L2 미스가 발생하여 주메모리로부터 이 데이터를 인출해와야 할 때, $15\%2=1$ 이므로 L2는 <홀수 큐>에 들어간다. 이때 실제로는 <블록 1>, <블록 3>, <블록 5>, <블록 7> 중 하나를 선택하여 저장하는데, 이중 어느 것을 선택할 것인가는 구현 및 각 시점의 저장 상태에 달려 있는 것이다. 즉, "HELLO"는 L2의 <블록 1>, <블록 3>, <블록 5>, <블록 7> 중 하나에 저장되고 어떤 방식으로든 MRU로 표시되며, L1의 <블록 1>, <블록 3> 중 하나에 저장되고 MRU로 표시된다. MRU와 LRU를 표시하는 방법은 구현자의 선택이다. 본 발명은 큐라는 추상적인 아이디어만으로도 캐시의 동작을 설명하는 데 무리가 없으므로, 이후로는 각 캐시가 큐인 것처럼 기술하겠다.
- [0062] 도 6 및 도 7은 종래 기술 및 본 발명의 캐시 운영 방법에 따른 데이터 입출력의 일실시예를 각각 도시하고 있다. 편의를 위해 <짝수 큐>만 도시하였다. 데이터의 요청 순서는 2-4-0-4-8-4-6-4로 동일하다. 이 숫자들은 전술한 바와 같이 메모리의 주소를 의미한다. 이후 편의상 그냥 숫자로만 기술한다. 예를 들어 2라는 요청은 주메모리의 <블록 2>에 저장되어 있는 데이터가 요청되었음을 의미한다.
- [0063] 먼저 종래 기술을 도시한 도 7을 보면, 현재 L1의 <짝수 큐>는 4와 6을, L2의 <짝수 큐>는 4, 6, 8, 2를 각각 저장하고 있다.
- [0064] L1이 2라는 요청을 수신했을 때, 이 요청은 $2\%2=0$ 이므로 <짝수 큐>에 대한 것이다. L1의 <짝수 큐>에 있는지 살펴봤지만, 없으므로 L2에 요청한다. L2의 <짝수 큐>에 있으므로, L2에서 MRU로 변경되고, L1에 요청값이 반환된다. L1이 꽉 차 있으므로 L1에서 LRU인 6이 제거된 후, 2가 L1에 MRU로 저장되고, L1은 이 요청값을 반환한다.
- [0065] 위 첫번째 요청은 L1 미스, L2 히트일 때이다. 두번째 요청은 L1 히트일 때이고, 세번째와 네번째, 다섯번째와 여섯번째 요청은 각각 L1 미스이면서 L2 미스인 경우와 L1 히트인 경우가 번갈아 반복되고 있다.
- [0066] 문제는 일곱번째 요청에 의해 다시 L1 미스이면서 L2 미스가 발생했을 때이다. 주메모리의 <블록 6>에 저장되어 있는 데이터를 요청받았지만 L1에도 L2에도 없으므로 주메모리로부터 데이터를 가져와 넣으려 했는데, L2 큐가

꼭 차 있다. 따라서 LRU를 제거해야 한다. 이때 L1 캐시와 L2 캐시의 일관성을 유지하기 위해 L1에서도 해당 데이터를 제거하지만 이는 L1에서 MRU인 데이터이다. 전술한 바와 같이 MRU는 가까운 미래에 다시 참조될 가능성이 가장 큰 데이터이므로 이렇게 어쩔 수 없이 제거될 경우 캐시 시스템의 성능이 저하된다.

[0067] 그러한 일이 여덟번째 요청에서 바로 일어난다. 방금 제거된 MRU 데이터가 요청된 것이다. L1에도 없고 L2에도 없기 때문에 메모리까지 갔다와야 하므로, 전술한 바와 같이 지연 시간도 길다.

[0068] 반면 도 8을 보면, L1, L2간 메시지 주고 받음 없이도 계산을 통해 다섯번째 요청을 처리하는 과정에서 미리 이러한 사태를 방지하고 있다.

[0069] 상위 메모리인 L1에서는 큐의 데이터들이 자신의 시프트 카운터를 함께 저장하고 있다. 첨자로 나타낸 것이 시프트 카운터다. 앞에서 도출한 바와 같이 본 실시예에서 이 시프트 카운터의 임계값은 3이다($(n*y)/x-1=3$).

[0070] 첫번째 요청에 대한 처리 과정을 살펴보면, L2로부터 가져온 2를 L1에 MRU로 넣을 때 0으로 초기화된 시프트 카운터를 같이 저장한다는 점(S210)과 나머지 데이터인 4의 시프트 카운터에 1을 더한다는 점(S220)이 도 7과 다르다.

[0071] 두번째 요청처럼 L1 히트일 때는 해당 데이터가 MRU로 변경될 뿐 나머지 항목들의 시프트 카운터에는 변함이 없다(S108).

[0072] 도 7에서처럼 L1 미스이면서 L2 미스인 경우와 L1 히트인 경우가 번갈아 반복된다. 이에 따라 다섯번째 요청 처리 과정에서 L1 큐에 시프트 카운터가 임계값인 3에 도달한 데이터가 발생한다(S230). 해당 데이터의 시프트 카운터를 다시 0으로 초기화하고(S240), L2에도 통보하여 해당 데이터를 MRU로 바꾸게 한다(S250).

[0073] 이후 요청에서도 도 7과 달리 L1의 MRU를 L2와의 일관성 유지를 위해 희생시켜야 하는 일 없이 이제까지와 마찬가지로 동작한다.

[0074] 이렇게 본원 발명은 L1, L2간 트래픽을 많이 발생시키거나 처리 지연을 늘리지 않고도 간단한 계산만으로 L1의 MRU가 희생되는 것을 막고 있다. 이를 위해서 필요한 것은 L1의 각 블록에 대한 시프트 카운터 뿐이다. 그런데 시프트 카운터를 저장하기 위해서는 각 블록당 버퍼가 필요할 것이다. 시프트 카운터의 최대 임계값은 $(n*y)/x$ 이므로 버퍼의 크기는 이에 대한 2를 밑으로 한 로그 값이다. 즉, 각 블록 당 $\log_2((n*y)/x)$ 비트 만큼의 버퍼가 필요하므로, 총 $(x/p)*\log_2((n*y)/x)$ 비트의 추가 버퍼가 필요하다.

[0075] 전술한 본 발명의 설명은 예시를 위한 것이며, 본 발명이 속하는 기술분야의 통상의 지식을 가진 자는 본 발명의 기술적 사상이나 필수적인 특징을 변경하지 않고서 다른 구체적인 형태로 쉽게 변형이 가능하다는 것을 이해할 수 있을 것이다. 그러므로 이상에서 기술한 실시예들은 모든 면에서 예시적인 것이며 한정적이 아닌 것으로 이해해야만 한다. 예를 들어, 단일형으로 설명되어 있는 각 구성 요소는 분산되어 실시될 수도 있으며, 마찬가지로 분산된 것으로 설명되어 있는 구성 요소들도 결합된 형태로 실시될 수 있다.

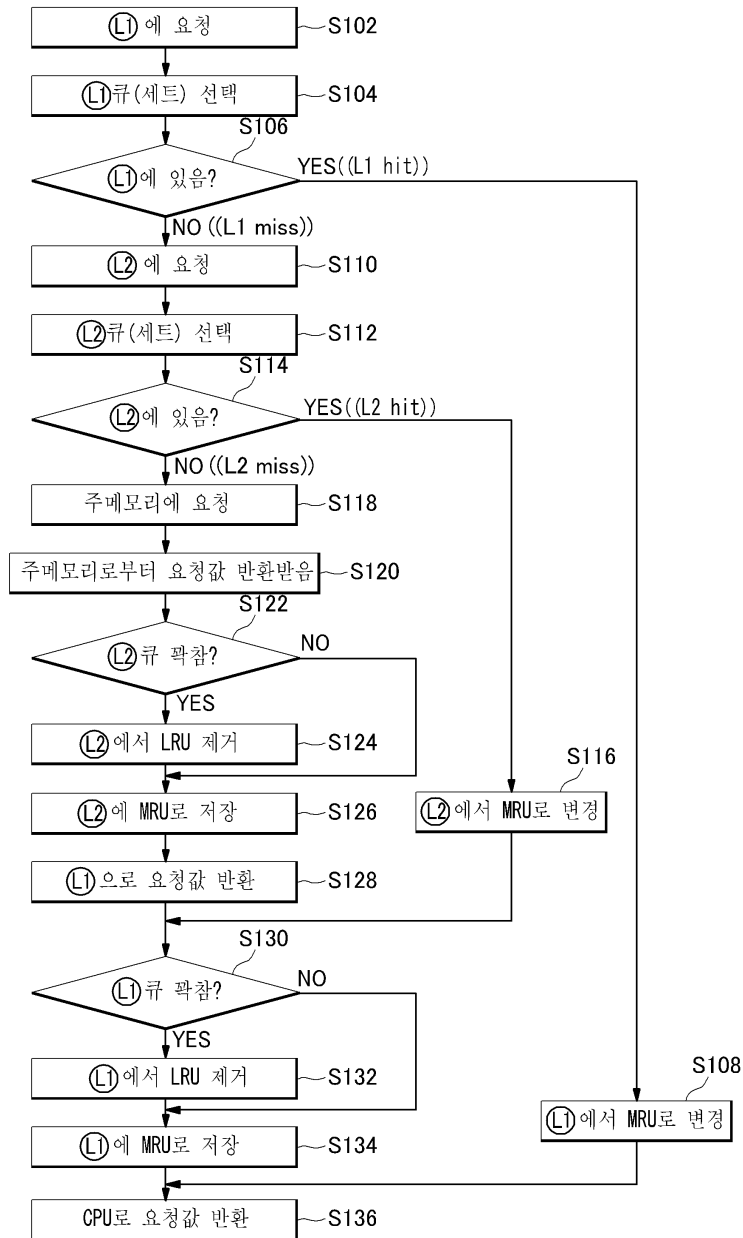
[0076] 본 발명의 범위는 상기 상세한 설명보다는 후술하는 특허청구범위에 의하여 나타내어지며, 특허청구범위의 의미 및 범위 그리고 그 균등 개념으로부터 도출되는 모든 변경 또는 변형된 형태가 본 발명의 범위에 포함되는 것으로 해석되어야 한다.

부호의 설명

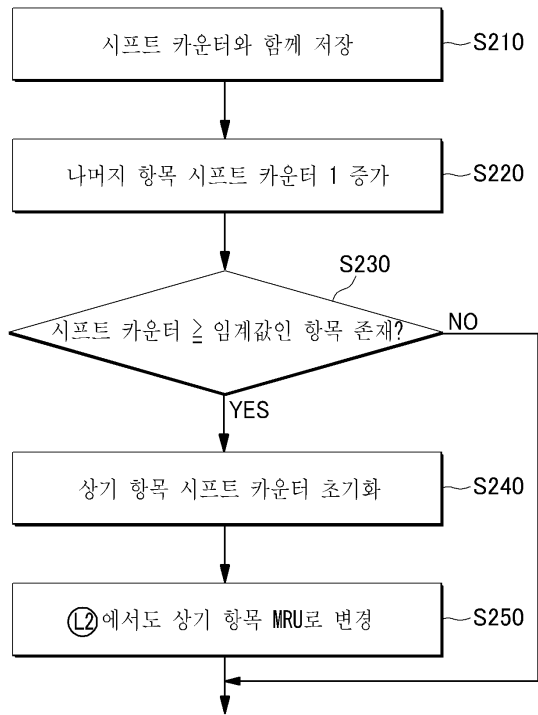
- [0077] 100: 상위 메모리
200: 하위 메모리
300: 한 수준 더 하위 메모리

도면

도면1



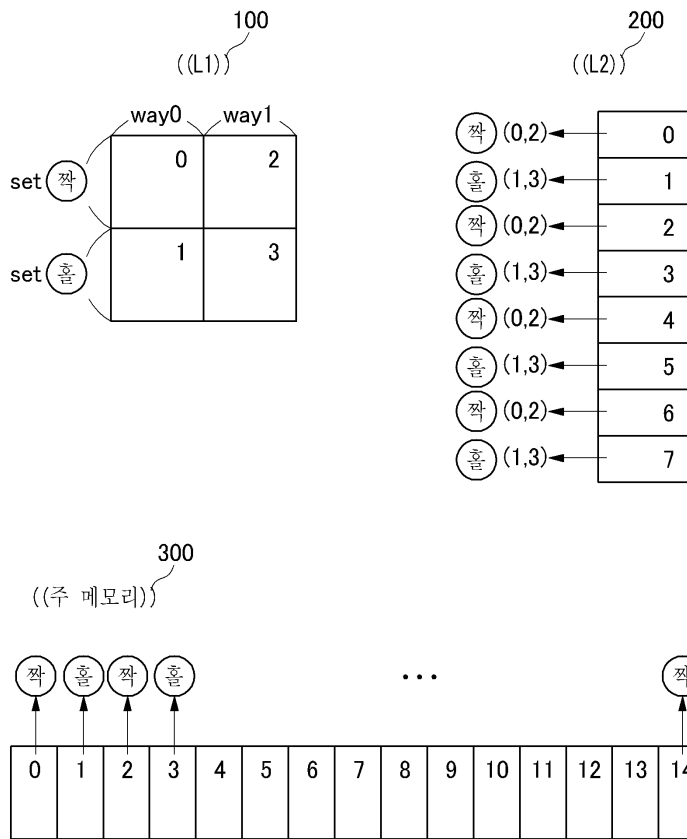
도면2



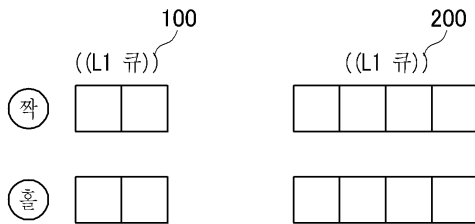
도면3

	블럭크기 (바이트)	총용량 (바이트)	웨이 수 (개)	세트 수 (개)	큐 개수 (개)	큐 길이 (항목수:개)
L1	p	x	n	$\frac{x}{n \cdot p}$	$\frac{x}{n \cdot p}$	n
L2	...	y	...	...	$\frac{x}{n \cdot p}$	$\frac{y}{(\frac{x}{n})} = \frac{n \cdot y}{x}$
메모리	...	z	...	...	...	...
L1	16	32K	2	2^{10}	2^{10}	2
L2	...	2M	...	...	2^{10}	2^7
L1	1	8	2	4	4	2
L2	...	32	...	...	4	8
L1	2	8	2	2	2	2
L2	...	16	...	...	2	4
L1	1	4	2	2	2	2
L2	...	8	...	...	2	4
메모리	...	16	...	...	...	...

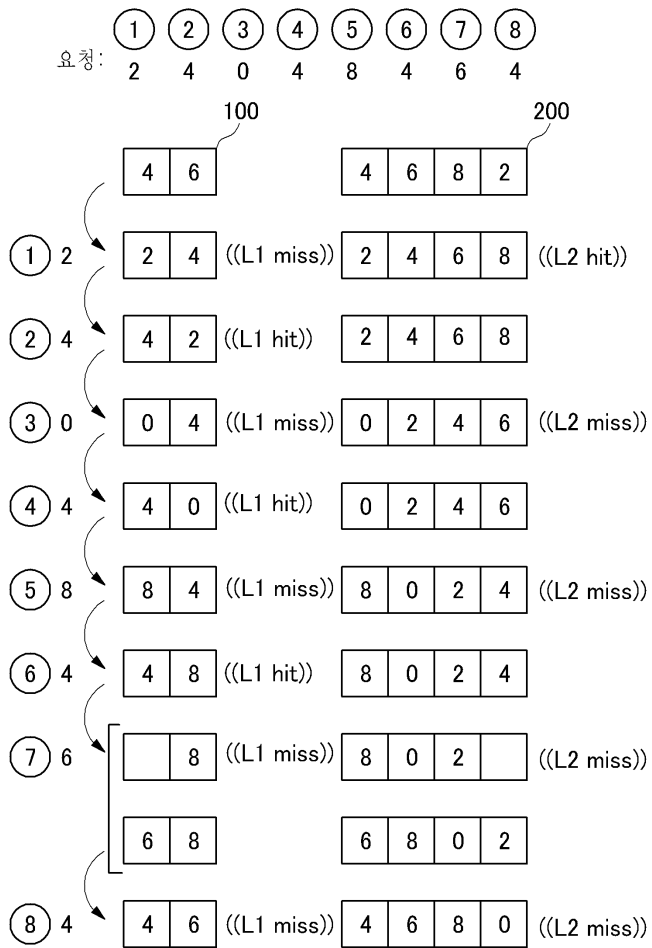
도면4



도면5



도면6



도면7

